

# EJB's 101 Damnations

By

Dino Fancellu  
Robin Sharp  
Matt Stephens

[www.bad-managers.com](http://www.bad-managers.com)

[www.javelinsoft.com](http://www.javelinsoft.com)

15 March 2002

Copyright © Matt Stephens 2002

This article may be freely distributed, under the following conditions:

1. The article is not modified in any way.
2. The article and this Acrobat file are left intact, i.e. nothing is left out (including this title page).
3. Distribution of the article is strictly for personal and not-for-profit use (e.g. forwarding it to a colleague). To discuss distribution for other purposes (e.g. magazine publication), please contact Matt Stephens at: [matt@bad-managers.com](mailto:matt@bad-managers.com)

## ***Table of Contents:***

1. Introduction.....	3
2. Conceptual Issues (High-Level “Damnatians”).....	4
3. Design.....	6
4. Development .....	10
5. Deployment.....	14
6. Run-Time .....	16
7. Knock-On Effects.....	18
8. Conclusion.....	20

## 1. Introduction

This is the tale of 101 Damnations. Sadly, it's no Disney story - in fact it's more likely to have been pulled from the pages of the Brothers Grimm.

Whilst writing the EJB modules for our [JGenerator](#) product, we started punting an email back and forth containing all the issues we had with EJB. And there were many. Before we knew it we ended up with 101 howlers.

In this article we present an edited version of the email. It isn't a complete list and there may be some overlapping issues. If there is enough interest we will turn it into a more formal analysis.

Before you flame us for chasing after your sacred cow with a big stick, all three of us have belonged to the "pro-Java" lobby since Java was in its first beta, and we still do. We have evangelised Java at every company we have worked at. However, EJB just presents so much that is wrong and messed up that we felt we had to speak out.

The 101 issues are divided into the following categories:

- Conceptual
- Design
- Development
- Deployment
- Runtime
- Knock-On Effects

And finally there's a conclusion, containing some heartfelt suggestions for improving the EJB spec.

We want Java and J2EE to succeed. We want it to remain the platform for developing server-side applications. We just hope that somebody at Sun is paying attention...

### Why Have We Written This Article?

Some common feedback has been: "You don't have to use EJB..."

We know that. That's why we say EJB is not the same as J2EE. But tell Sun that. Tell their marketing. Tell the world, the server vendors, the hardware vendors, the body shops.

I can say that some food is damaging to your health. You tell me that people don't have to eat it. Does that mean I shouldn't have told people?

Our article serves as a warning, a lighthouse on the rocks. Pointing out that people don't have to crash onto the rocks hence the lighthouse does not need to be there seems violently stupid.

## 2. Conceptual Issues (High-Level “Damnations”)

EJB's choice is Hobson's Choice. If you don't like EJB's locking, caching, performance, failure or persistence mechanisms, then write your own, says JavaSoft.

101 Damnations to you.

"Thomas Hobson, a seventeenth-century liveryman in Cambridge, England, told every customer he could have any horse he wanted as long as it was the one nearest the door. Hobson's choice should not be used in the context of dilemma or mere indecision. It is a choice between what is offered and nothing."

*Source: Communication World December 1992*

On to the Damnations:

1. EJB represents a radical departure from the Beans model. Beans was based on the (proven) component model also used in Delphi. This means that for projects that are migrating to EJBs, much of their code base is no longer valid.  
  
Very little effort was made getting EJB to naturally extend the Beans specification (remember Java Beans are not just about GUI widgets).  
  
For example, there is no mention of Events in the EJB model. Enterprise Beans don't have property change events or vetoable events, or indexed properties. The specifications focus almost entirely on the server side specification and hypothetical roles.
2. A rigorous set of examples and design goals for application programmers has been missing from the start. Instead the focus has been on remote objects and pessimistic locking, rather than the high performance and soft locking required by most Internet applications today.
3. EJB is based on an Object Pooling optimisation. Optimisations should not be incorporated into the design, but left for vendors to decide and compete on. The logic of basing a system on saving a resource as cheap as memory seems absurd.
4. Sun's recommendation to "Use Container-Managed Persistence when you can" is contrary to our experience. I am sure we're not alone in that CMP seems to be useless for complicated applications.
5. The EJB specification makes no reference to other well known distributed system design references - for example the ISO Basic Reference Model of Open Distributed Processing 1992.

Most seminal communication specifications make reference to the ISO Open Communication Models. Either there is a lack of theoretical grounding, or perhaps the designers didn't think it was worth mentioning.

The above-mentioned ISO ODP reference model has several goals for a distributed system. Many of the important goals have not been met - in particular (points 6-9 as follows):

6. The EJB spec doesn't address access transparency. Local and remote object definitions are not transparent, as you inherit from a different interface and have to cast using the narrow convention.
7. The EJB spec doesn't address performance transparency. The performance solutions for EJB involve radical redesign of your code, such as session-based methods, value objects, entity bean serialization or client side caching.
8. The EJB spec doesn't address migration transparency. If any object moves, the client must throw away the remote or local interface and get a new one.
9. The EJB spec doesn't address Concurrency transparency. There are no clear sets of locking strategies to suit different application programming tasks. The application and EJB programmer is left to handcraft locking strategies.

### 3. Design

Three years on from EJB 1.0 and still not ready for prime time. Design issues cloud development work.

Compare this with SQL, which was not even the best query language at the time it was introduced, but is now used almost universally. SQL as the client-facing interface did not try to tell database vendors how to implement internals.

The EJB specification is monolithic and causes lots of dependencies for EARs. The result is that the beans are brittle. Below, we provide some examples of why this is the case.

In addition, please bear in mind that not all of these items are outright criticism; remember this article is also a "wish list" of items that we feel would improve the EJB spec.

This section focuses on the design issues that we believe are preventing EJB from gaining greater acceptance.

10. The only justification for EJB is on large projects, yet EJB makes those large projects larger still, and yet more unwieldy. You need to scale to get EJB to perform.
11. Role based class/method security is rarely useful. Data security is far more important.
12. EJB Query Language is another language to learn. This is reminiscent of the situation with WML, where an attempt to write a simpler language than HTML failed because they didn't just simplify HTML. Instead they started from the ground up, having decided erroneously that they could do a better job themselves.  
  
EJB Query language doesn't support many SQL operators, (e.g. subqueries, IN, ALL, ANY, String comparison, ORDER BY, GROUP BY).  
  
There is a good critique of EJB-QL [here](#).
13. The specification does not provide for proper handling of views or read only tables. Back in the real world, read only tables and views are very important.
14. EJB only handles first class objects (i.e. beans that have primary keys). It would be nice if it also handled second class objects (do not have primary keys), where they have a one-one relationship with a first class object.  
  
For example, a Cash object might contain a currency and an amount. Using CMP Entity Beans you would be forced to have a Cash table. A better design would be to automatically map the Cash.currency and Cash.amount into the parent table's columns.

15. No proper handling of validation (e.g. String lengths). For a specification whose main purpose is reading and writing to a database, this is a surprising omission.
16. No proper handling of Enumerated Types. Enumerated Types represent fixed sets of data, such as Sex, Industrial Classification, or Marital Status. It should be possible to use them as properties on Beans.  
  
It has been argued that this is more of a Java issue, i.e. a deficiency in the Java language rather than EJBs per se. This is not necessarily the case though. For example, our [JBeans](#) spec includes Enumerated Types defined via a standard interface and an EnumeratedType abstract class.  
  
Adding Enumerated Types to the Java language would be better; but the EJB spec would definitely benefit from a similar addition.
17. EJB doesn't make a clear separation between the components and the machinery that is supposed to persist it. For example a bean can remove itself, which assumes a bean is always persisted and it must always be tied to a persistence mechanism.
18. Entity Beans bind the business logic with the persistence mechanism. Hence the need for Data Access Objects. If the persistence was orthogonal to the beans this functionality would have been accommodated more naturally.
19. Optimistic locking is not accounted for. The EJB spec assumes either pessimistic locking or not locking, or roll your own. There could have been a locking interface; or a way to specify a logical concurrency mechanism like isolation levels.
20. Most 'fast' EJB patterns seem to involve using sessions that go straight to the metal. Half the time is spent navigating, so half of the system ends up being rewritten.  
  
Entity beans only appear to be useful in single entity transactions, but not where the entity bean needs to know about its dependent beans. When you save a bean, the graph of dependent beans should also be saved - but it doesn't work like that.  
  
For EJB you can specify cascading deletes, but not combinations of creates, updates and deletes. To save a graph of beans you must create your own client side User Transaction and call each create on the home, set on the bean and remove on the bean. The beans should understand what has been changed in the graph and update the database accordingly.  
  
If we were "putting the boot" into EJB, we would suggest that this means EJB is only half complete.
21. There is no meta-data for business, i.e. no meta level to describe your business. Just code.
22. EJB tries to address both the client side interfaces and the service provider interfaces. Really as a client we should only care about the client side interfaces. This would make EJB open to different implementations.

23. No standards for writing Session beans as Beans. Yet this approach is one of the most talked about on the web.
24. There is no concept of dependency, so that a bean is stored, removed or updated at the same time.
25. There is no concept of whether the primary key is generated externally by the application, internally by the container, or automatically by the database.
26. EJB is designed to handle high-transaction units of work in a distributed environment. A very small percentage of applications need this. When we define high-transaction usage we mean thousands, not hundreds. Most users need only tens a second. The EJB spec also neglects applications that need to return static data, or real-time data. There are not designs in place to return these objects.
27. No standardised integration with content management tools (other than "roll your own" solutions via JSP, for example). For large web sites, it isn't clear how these would integrate.

This is more of an issue with J2EE as a whole than EJB by itself.

28. To attach enterprise development without going to a higher level, without approaching it with a blueprint, is like trying to dig a tunnel through a mountain with ever so slightly sharper spoons. They shouldn't be working on this level.
29. Sun doesn't have a joined up strategy for persistence, i.e. JDO, JDBC, Java Blend, serialisation, XML, EJB without any common interfaces. This is similar to the case with GUIs, i.e. Swing, AWT, MIDP, and web front-ends which they are trying to address with the mysterious JavaFaces. This gives the impression that the various design groups at Sun are out of control.
30. ACID is not always wanted. For example, a search engine wants high performance, availability, in exchange for accuracy.

As with many of the issues raised in this article, the obvious answer appears to be: "So don't use EJB for this type of work." Unfortunately, we believe that this blinkered approach seriously limits EJB's usefulness. Why should it be limited to such a narrow field of use, simply because of some erroneous assumptions made early on regarding the sort of features that *all* enterprise applications would surely want?]

31. Most web apps are not very transactional. EJB comes from a TP/MTS type background. Perhaps it should have been called TJB, Transactional Java Beans, so that people knew where it belonged. As it is, EJB co-opts areas that simply don't belong to it. It's a baroque framework for transactional Java beans, which tries to wear the clothes of the enterprise emperor.

EJB is very transactional - it sucks when it comes to high performance queries, read-only work.

32. EJB hasn't really addressed the issue of efficient access to remote objects. Effectively you have hand code. It would have been better to have client

side beans where vendors could have generated different options. At the risk of plugging our own product, JGenerator allows you to do this.

33. There is no clear design for returning Relationships between beans. Should you return a primary key (and look it up on the client)? Or return a handle, which does the look up but needs Bean managed code, or return the related beans remote interface which in turn means referencing the remote bean from the server?

Container managed relationships can only exist between beans whose homes are on the same server. This makes it impossible to deploy an enterprise system where one bean references a bean in another database automatically. For example, several product databases may reference a central customer database. There is no way to automatically generate a `getCustomer()` method on a product. Not very Enterprise.

34. It's difficult for a bean to tell the server when it is dirty, or read-only. This is supported in some servers, not in others. This should become part of the specification.

## **4. Development**

One application speaks volumes about the state of EJB development: Petstore. Even Microsoft have picked up on this supposed example of correct EJB design, and trounced its design and performance (see the fallout from this episode at [TheServerSide](#)).

Sun's response was that Petstore is not supposed to be used as a model of speed and efficiency. Why not? If EJB was a brilliant design, this would never have been an issue.

35. Use of CMP beans locks you very much into the vendors' proprietary mapping technology. Beware of the apparent ease of use that CMP tries to offer.
36. The EJB Spec imagines the role of a container builder and a server builder. This assumption that there would be separate container products that would run in the application server is incorrect. Hence the service puts unnecessarily tight constraints on state control. This should have been left up to the vendors.
37. Development is very slow. For example, the development cycle is tens of minutes, rather than seconds, simply to change a small value in a JSP. This is a huge project killer. Increased development time hardly helps with 'Internet time' projects.
38. There is no concept of a design time state, like in the Beans model that allows for a quick development cycle.
39. EJB represents a huge learning curve. The EJB spec is supposed to help, but seems to hinder. It's very complex. Even for the most experienced engineers and architects, getting all the niggling details correct makes development painfully slow.
40. Many sites do not need EJB complexity, yet people use EJB for the sake of fashion, or getting skills on their CV. Too many people are happy to sacrifice the future of their company/project, in order to get 'sexy' skills.
41. Petstore. No databases are used to stress test the design or implementation. It's as badly thought out as sending pets through the post.
42. Petstore is too big to be a quick demo test case.
43. Petstore is too small to stress test the memory.
44. Petstore involves too little data to stress test scalability.
45. Petstore is too homogeneous to stress test Code Generators. This is probably a slightly unfair complaint to make, as Petstore was obviously not designed with automated code generation in mind. However, code generation is an increasingly important aspect of EJB design and development as it removes the drudgery of churning out endless beans to the same design pattern.

Therefore, we would argue that the J2EE reference application should reflect the manner in which an increasing number of EJB projects will be written. In this regard, the problem with Petstore is that it is very samey all the way through, and the database doesn't do enough "weird" things. For example: the tables don't contain all the data types, there are no 3 way primary keys, and no views or read only tables.

The result is that code generating Petstore doesn't prove very much.

46. Vendors all supply their own versions of Petstore, where each seems to be mostly handcrafted.

There should only be one version of Petstore as a container managed bean that is fully working. At the moment every vendor has to create their own CMP to get it working.

47. There is no way of knowing when set property values are done. Explicit transaction demarcation is done at the property level, not at the Bean level. If there was a store method on a home/session this would do this.
48. There are no standards for optimising Entity Beans, serializing the whole bean or sending property sets.
49. One way to fix the previous two problems would be to introduce "smart stubs" \*. Instead however, the programmer is expected to write straight to the wire for every property get or set, or to use facades, dependent objects etc. That's an awful lot of extra code, especially when it conceivably affects every single entity bean in your application.

\* Smart stubs are a concept that CORBA came up with. A client side CORBA stub would have another stub subclassed (and could be delegated from). So the client side programmer thought they were going straight to the remote stub, but the application programmer had the opportunity to 'fiddle' with the call before it went.

Subclassing meant you could only do a little bit of fiddling, but delegation allows you to completely alter the RMI method call, so that you might write to the client a sequence of "set" method calls, and all the time it would be storing the set values in a property list, instead of shoving them over the wire every time.

Then, when you call the commit transaction method, all the sets that you have done on all the objects in that transaction are bundled up and sent in one message. Conversely all the properties can be pulled over when you call the first get, so you're not getting every tiny property every time you call a get method.

At Javelinsoft, we do this in our CJBs ([Client Java Beans](#)) by serializing the Entity Bean from the EJB server when we make the first get call, and filling it up and squirting it back when we say save on the Home.

50. Stateless session beans are supposed to be the most scalable, yet they are not much more than RMI. However we still need to jump through all the hoops to get them working.
51. It isn't easy to put EJB in Applets. There is no concept of a lightweight interface.

52. EJB does not have count methods optionally automatically generated. This feature is used in many applications I have worked on to save loading objects into memory.
53. Non web app client access to EJB is vague, and inconsistently supported. Sometimes an application server will supply you with a jar to include in your application to get access to the EJBs; sometimes you just have to figure it out yourself and end up including several megabytes of jars.
54. The spec for EJB cross beans mapping is constantly under change, and inconsistent across servers. It's also not flexible. The result is that you cannot easily port from one application server to another.
55. We need to use value objects to get any real performance, else each get/set makes an RMI call. Why was this concept of local stubs or property sets not put into EJB? Performance should not be an irrelevance.
56. The EJB Compliance Tests do not appear to include proper examples that rigorously test all scenarios. Because the CMP and BMP are different for all vendors, it would be nice to have some examples that have say 100,000 rows being loaded and used, or 100 rows containing all the data types being tested.

For example, in JDJ volume 7 issue 1, an HP engineer reveals why the CTS is so important vs the reality of the process that they endured with HP-AS.

How is it possible to be 'compliant' yet full of bugs? Answer: [bad tests](#).

So yes, we need tests, much better tests. But these synthetic tests, even if they weren't so bug ridden, would still only test some synthetic criteria.

If instead they simply said, "let's try to use EJB to do business in these scenarios" and picked 5 or 6 hard business areas they'd do much better than all this synthetic design.

If we examine Java's history, we see that the same was also true for Swing. The specification would have advanced much faster if they had simply said, "let's do Office in Swing". You can only abstract from the concrete, yet Sun engineers again and again try to abstract from their imaginings, guessing what might be cool, or pure, as opposed to what people need. What's the point of writing software to meet future needs if it won't even meet your current ones?

Another problem is that the tests are only done on toy examples, and there are no theoretical underpinnings. Therefore, you are never sure when EJB is going to let you down. At least with Java and SQL we know there is a rich history and/or theory so we can make measured predictions of what we can do and how the system will perform.

57. EJB wizards are often shaky and dangerous. They end up creating code that simply won't work. So you waste time thinking it must be your own problem somewhere. For example, the RI (Reference Implementation) 1.3 container entity bean wizard misses out "throws CreateException" on `ejbCreate`.

58. The RI seems to have had 0 to little Q.A. For example, 1.3 RI code generation tools simply produce wrong code. Their entity bean managed findByPrimary would never work. This is of no help to anyone learning EJB.

As the RI is the "Reference Implementation" this is very bad. No one expects the RI to be fast, but we do expect it to be a model of compliance and correctness.

## 5. Deployment

Deployment remains one of the most frustrating and time-consuming aspects of EJB work. This section explores some of the reasons why, and suggests what might be needed to improve the develop/deploy/test cycle.

59. The WAR/JAR/EAR model is horrible for development and deployment. The deployment spec is opaque, using the [Russian Doll](#) metaphor.
60. EJB deployment is incredibly fragile. Miss setting a property and the container won't use a sensible default. Of course this depends on the container, but mostly they simply throw a vague error message and refuse to continue. This is just plain unhelpful - "jobsworth", even.  
  
The servlet world has it easy in comparison - e.g. the "invoker" servlet executes anonymous servlet classes that have not been defined in a web.xml file (usually maps to /servlet/\*).  
  
In other words, servlet engine writers go to special efforts to make deployment easy. Easiness is what it's all about. EJB containers don't have the equivalent of a default web.xml file that they can inherit default properties from.
61. There is no standard way to deploy an EJB application. The details change with each server, and (worse still) with each revision of each server.
62. EJB needs to be more IDE-friendly, particularly when it comes to deployment.  
  
Deployment is not always opened up to tools, e.g. RI 1.2.2/1.3 uses batch files which call Java. Its deploy/package code is not meant to be called by tools. The RI uses the broken concept of current directory, i.e. must be called from a DOS window, not from an IDE VM.  
  
Hopefully Sun's upcoming Deployment Specification will fix this problem. The spec needs to be such that it can be picked up and used by IDEs.
63. The EJB spec has too many fictitious roles. In reality there are container/server providers, and application providers who develop, assemble, deploy, and administer the EJB server. The EJB spec should reflect this reality.
64. There is no such thing as cross platform EJB. Each platform has its own horrible bugs. Escape one system to fall prey to another.
65. Developers have to repeat the same meta-data in the code as SQL, as client properties as server properties, as XML attributes.
66. Container managed EJB often requires you to specify a lot of what would be in your code, for example the SQL.
67. Administration is different with every server. There is no standard for administering the deployed EJBs.

68. XML is all about interpretation. Different vendors interpret the XML differently. For example (on the servlet/JSP side), web.xml's <welcome-file-list>. Some see this as 'look for these files in this order', others as 'these files should be here'.

Of course this sort of misunderstanding just can't be helped sometimes, but is included here as something to be wary of - it's an example of how ambiguity can creep into a seemingly bullet-proof spec and cause standards to fragment.

69. The specifications are continually changing. That's okay, but when new versions cannot run old applications without major changes, then we have a problem.

70. The server vendors have graced us with some remarkably vague and misleading deployment tools, e.g verifiers that spout unhelpful error messages, or pass classes that then fail on deployment.

71. Loose coupling between the bean and interface. The XML is not strongly typed, i.e. too much room for error. Very little use of reflection to work out class relationships.

72. Deployment of code, in order to test it, is a horribly complex, click-laden process. As well as involving many stages, there seems to be no proper way to automate it, especially if you are attempting to target many different EJB servers.

73. The basic ejb.xml, web.xml and application.xml descriptors are not sufficient to deploy. The result is that further descriptor information must be written using app-server specific XML.

This means that every ear must be changed to be deployed. It would have been better to have defaults set up so that a minimally defined EJB would deploy and run without modification.

74. The XML descriptors use ambiguous names - for example, displayName isn't just the display name, it's used as the reference name.

75. An EJB ear should be able to be deployed on ANY compliant EJB server, with no changes to the ear.

It should expose and advertise the environmental items that can be changed (like war mount point), and those that MUST be changed (like database connections).

However, there should be no requirement on the ear developer to produce X different versions, for all the servers out there. This seems to be the current state, which is madness.

76. The Validator passes descriptors which later fail on deployment. For example (using Catalina/Tomcat in RI 1.3), try creating a war file with multiple entries with the same name. The war is passed on verify, but of course fails to deploy.

A connected problem is that once deployed, there is no way to automatically test that your beans have actually deployed.

## 6. Run-Time

Once you have developed your Enterprise Beans and web apps, packaged them into their respective Jars, Wars and Ears, deployed them to your favourite gronky EJB server, fought with the deployment tools...

... edited the auto-generated deployment descriptors to make them work, and then actually seen your JSP page display some data, and rejoiced that the Gods of EJB have allowed you this small victory... then you might think that you are home dry.

Unfortunately, there are still the Runtime issues to be grappled with.

77. The real bottleneck is data access and transactions, of which the average database can only handle about 50-100 a second. EJB needs to take on board replication server duties in order to really speed things up.
78. There is very little scope for run-time optimisation. The EJB specification contains no standard flags or extensions for improving performance. For example, WebLogic has used a 'dirty' flag to help determine if the object should be written down to the database. Despite being a well-known extension with well-understood semantics, this didn't make it into the latest specification.
79. EJB performance is very slow, and uses lots of resources. It's costly to scale. The "fetch my primary key" query model suggests that the designers did not understand how relational databases work:  
  
The EJB queries return a list of primary keys. Each of these primary keys are then used to perform another query to the database. This operates in a way contrary to the way relational databases are optimised, i.e. on sets of rows. The whole philosophy behind EJB object pools is to re-use objects, and not cache them. This relates to Damnation #3 - that EJB is designed to preserve memory, when memory is not a scarce resource - in fact it's a dime-a-bucket at the moment.
80. EJB cannot scale. Because your entity beans need to be synchronised across all the servers that they are deployed on, the amount of network traffic multiplies.
81. The Reference Implementation (RI) is fragile, easy to break. 1.2.2 simply broke on perfectly valid code. It also isn't backward compatible - an ear that worked on an earlier release simply doesn't run on a later release.  
  
The specification was not specific enough to guarantee certain behaviour.
82. EJB servers which have their own Servlet/JSP engine often get it wrong, e.g. WebLogic 5.1 needs to get to SP7 at least to be even mildly reasonable. Servers like Orion and HP-AS still have issues with basic functionality. So even if the EJB side is fine, your web apps will malfunction.

83. Black box. When your deployed system goes slow you can't work out why unless you have the container source.
84. An EJB server is harder to build than a servlet engine. Hence they tend to be far more fragile and mad, which is hardly a good base for an enterprise application.
85. Sun said that to be 2.0 compliant an EJB app-server would also have to support 1.2 compliant EJBs. But their own reference implementation does not support 1.2 compliant beans, because the Sun specific 2.0 XML interpretation throws errors with 1.2 compliant XML.
86. Sun's J2EE compliance tests seem to let some pretty obvious omissions through. For example HP-AS doesn't handle nested archives (i.e. referencing one jar from another through a declaration in the Manifest file), which is part of the J2EE spec. A server can be officially branded J2EE, even if half its J2EE features are broken. The promise that they'll be made to work in the next service release doesn't quite cut it, and often ends up not happening.
87. Can Sun PLEASE focus on the RI, working around the clock, until it is 100% in line with the specification? Will we ever get to the state where the RI is in line with its goals, or will the spec always be aspirational?

Why don't Sun take an established open-source EJB server such as JBoss, and work on making that their reference implementation (or alternatively, open the RI out to Open Source, in the same way they did with Tomcat for servlets/JSP)?

## 7. Knock-On Effects

EJB is baroque: extravagant, complex and bizarre. As a language matures, its creators have to steer the designs so they don't end up with a house of cards.

Java is going through the same maturation process, yet EJB has been badly thought through for the developer.

This section examines the effect that EJB's baroque nature has had on the enterprise marketplace.

88. Sun's marketing confuses people. Often, journalists and managers alike will confuse EJB with J2EE. Java on the server does not necessarily mean EJB.
89. EJB exposes Java to attack, not from criticism, but from a better language. Not talking about C#, but a language based on Components and Stores instead of classes.
90. Where is the large-scale EJB usage? The thousands of downloads of J2EE, WebLogic, HP-AS, JBoss etc don't quite tally with the number of real-life EJB systems in use.
91. Where is the large-scale usage of EJBs? How many companies are really using EJB for massively scalable back-end systems?
92. The Sun hype pushing EJB will backfire. It reminds us of their spin machine pushing Applets on the client before the Java UI was ready for prime time. This had a detrimental effect on Java.
93. Where is the thriving EJB component marketplace 3 years from inception? The situation seems similar to what was said of C++. VB/Delphi/Java had and has loads of components, as it is so easy to write them.
94. The EJB specification was not focused on any applications. There should have been two or three big, complex, real world apps that needed to be expressed in EJB as a testbed.

It took Sun about 2 years to come up with Pet Cemetery, and even then, it proves nothing except that they don't have a clue. This same criticism is just like that of Swing - it took years to come up with proper test apps, e.g. Forte!

95. Warnings from Gartner Group:

"As components of application server technology, J2EE and Enterprise JavaBeans (EJB) are not the same thing. Most Java projects use Java Server Pages (JSP)/servlet capabilities and not EJB. Higher-priced application servers are designed to run EJB, yet they are using JSP/servlet capabilities instead."

"Companies have overspent about \$1 billion on application server technology solutions since 1998. Moreover, an additional \$2 billion may be wasted between now and 2003."

"Don't let confusion or hype push you to spend more than necessary."

96. Sun's main revenue stream is from hardware, not Java. Their profit driver is to sell hardware. This explains why EJB has not been reigned in and rationalised.

97. There is no real synergy between EJB and the rest of J2EE.

98. Most, if not all, EJB articles on the web seem to be about hacks, bugs, workarounds, optimizations, as opposed to glowing success stories. Where are they?

99. The creators of the EJB spec are in a privileged position. It's very easy to define huge, complex specifications and expect the industry to forever be trying doggedly to catch up with actual implementations of their mad schemes and "visions".

It's not so much an OO Utopia as a "Complexia". It would be much harder for them to try and implement their own specs (as they discovered with the RI mess).

The server vendors don't get a chance to make their servers useable or robust, as they're forever trying to catch up with Sun's latest promised feature-list made on their behalf. Then there's the even bigger problem of supporting legacy versions (for those vendors that can be bothered with such "trivial" notions).

100. Brave New World - this isn't just a comment on EJB, but on the state of enterprise computing:

Everyone expected the OO dream to materialise with interconnected, co-operating objects whizzing around global networks - instead we have gronky EJBs and labour-intensive EAI systems (e.g. BizTalk), much too complex for their own good.

101. The basic idea behind EJBs, and Sun's approach to APIs, is good. The principle of defining standard specifications for the industry to follow is a noble one. But something, somewhere, has gone terribly wrong.

## 8. Conclusion

EJB makes Java look bad. It gives the enemies of Java a stick to beat us with. Can it be fixed though?

Much of this article reads like an EJB "wish list". If only certain key issues could be resolved, J2EE would become unstoppable.

We don't seriously believe that Sun would hold up their hands and say "it's a fair cop, let's scrap EJB - let's hit the reset button and start again, having learned from our mistakes."

It is far more likely that they will produce "point releases" that gradually fix EJB, one minute problem at a time. Local objects are a good example. The problem is that rather than being fixed from the ground up, the EJB spec will grow, and become ever more complex as the server vendors must support the legacy specs as well as the jazzy new "fixed" specs.

As it is, here are a few more recommendations to "fix" EJB:

- That bridges be built between EJB and the Beans spec. To come up with client side stubs like [JBeans](#) Session that fit into the beans spec.
- That development is simplified so that 1-tier, 2-tier and 3-tier are seamlessly catered for.
- That deployment is simplified so that we programmers can have back our concept of a separate design time and run time (i.e. not having to jump through hoops to package everything up, just to test-run/debug the slightest change). The simple addition of a deployment interface that IDEs can use would help immensely.
- That the container side specs are revised so that persistence is decoupled from the business aspects, as this doesn't add any value for the developer.

EJB is not as type safe as it could have been, so servers need to 'simulate' type safety by going through a verification phase. Verification is not done at design or build time - the APIs could have been designed so that more type checking was made between the Remote Interfaces and the Server side beans.

Alternatively, some kind of XML meta-constraints could have been implemented to introspect the Java and match it to the XML.

More importantly, there is no specification for passing or failing the verification phase - but there is a test for passing J2EE. It would have been nice if they were one and the same thing, so applications would pass or fail verification consistently for different vendors.

Back when Java was released, James Gosling said that Java was C++

without the knives and the daggers (with reference to how bad references in C/C++ caused programs to fail).

Now J2EE introduces bad references to a new generation of developers, causing programs to fail. The knives and daggers are back. Or is it just the sign of an over-ripe language?

## The Final Word (for now...)

EJB is not the final word in distributed systems. After RPC and CORBA, it is the third major release of possibly six or more architectures.

Almost every EJB tool vendor offers a "value-added" framework, showing that it can't stand up on its own as an architecture. EJB was designed by server enthusiasts for server enthusiasts.

Marc Fleury, who is the founder and lead of the JBoss project, [said recently](#) (regarding the future of J2EE):

"We see J2EE becoming embedded in applications. The 'object poets' want to see distributed systems with objects talking everywhere, but it just doesn't work that way. We don't take the paternalistic view of big servers small clients, we take the view of small servers, with the J2EE stack embedded in everything."

Looking at the problems with EJB, the next major distributed system will be based around the client side requirements, and not the server side implementation. The interfaces will be between business and presentation logic, for example a more advanced version of the Beans APIs to include the concepts of Home and Session. The interfaces will be between organisations, for example a more advanced version of SOAP that will include interfaces.

It is worth reinforcing the fact that J2EE is not EJB. A point made recently by Rick Ross of JavaLobby.org:

"J2EE is not spelled 'EJB', any more than rectangle is spelled 'square'. ... 90% or more of current J2EE solutions don't use EJB."

The next distributed architecture must also conceal the persistence mechanism. The current Java product offering includes many persistence mechanisms (EJB, JDBC, JNDI, MIDP, Java Spaces, Java Blend, JDO, Serialization, XML etc.)

The next generation of distributed systems will give application programmers a choice of architectures. Give them the freedom to choose between the costs and benefits of each architecture on offer. EJB's choice is Hobson's Choice: either put up with our design, or handcraft it yourself.

EJB is just a fashion. It mostly doesn't suit requirements, and we're given Hobson's Choice. To quote from Moby Dick:

"Oh! Ahab," cried Starbuck, "not too late is it, even now, the third day, to desist. See! Moby Dick seeks thee not. It is thou, thou, that madly seekest him!"